加微信：642945106 发送“ 赠送 ”领取赠送精品课程
发数字“ 2 ”获取众筹列表

05 | 案例：落地SLO时还需要考虑哪些因素?

2020-03-27 赵成

讲述：赵成
时长 14:16　大小 13.08M

你好，我是赵成，欢迎回来。

前面几节课，我们按照层层递进的思路，从可用性讲到 SLI 和 SLO，再到 SLO 所对应的 Error Budget 策略。掌握了这些内容，也就为我们建设 SRE 体系打下了一个稳固的基础。

今天，我用一个电商系统的案例，带着你从头开始，一步一步系统性地设定 SLO，一方面巩固我们前面所学的内容，另一方面继续和你分享一些我在实践中总结的注意事项。

案例背景

我先来给你介绍下电商系统案例的基础情况，框定下我们今天要讨论的内容范围。

一般来说，电商系统一定有一个或几个核心服务，比如要给用户提供商品选择、搜索和购买的服务等。但我们知道，大部分用户并不是上来就购买，而是会有一个访问的过程，他们会先登录，再搜索，然后访问一个或多个商品详情介绍，决定是放到购物车候选，还是选择物流地址后直接下单，最后支付购买。

这条从登录到购买的链路，我们一般称之为系统的**核心链路**（Critical Path），系统或网站就是依靠这样一条访问链路，为用户提供了购买商品的服务能力。

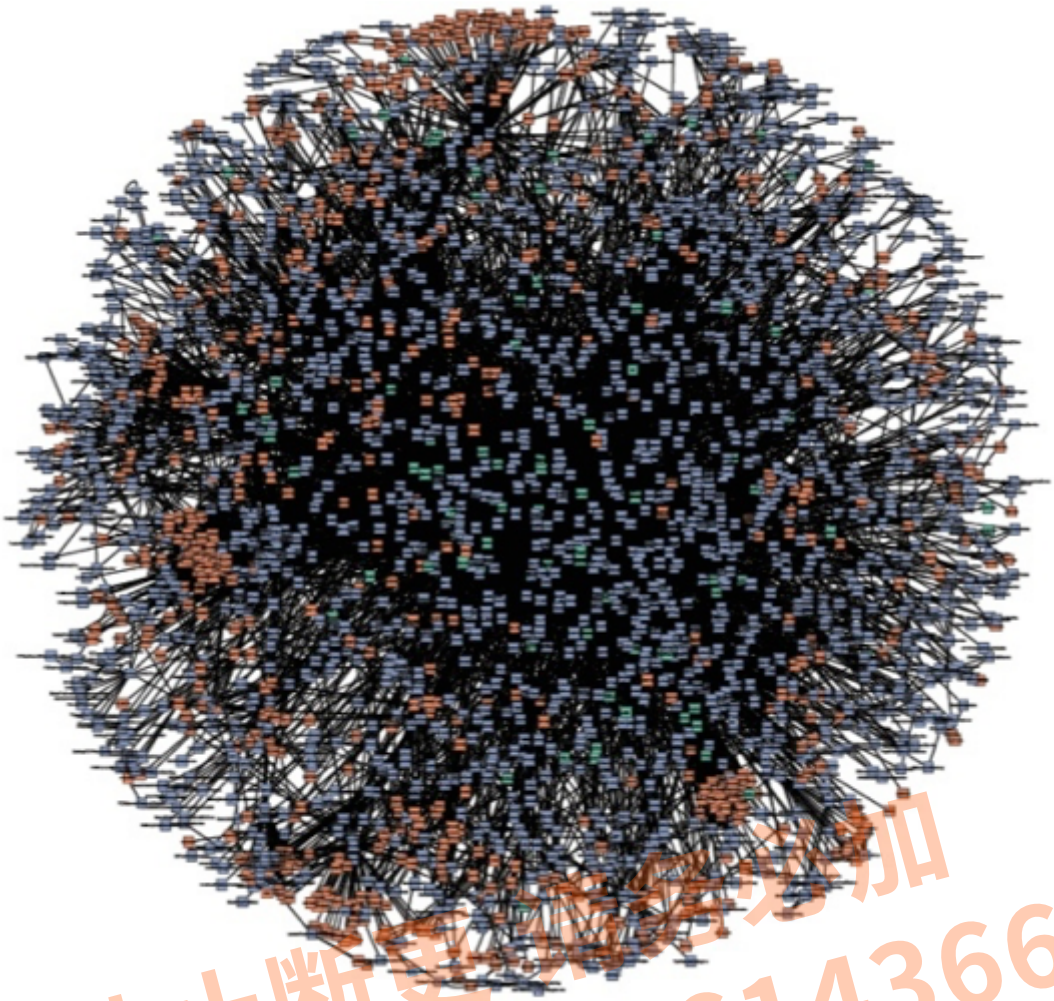
至于电商系统的其它页面或能力，比如网站政策、新手指导、开店指南等等，这些对用户购买服务不会造成太大影响的，相对于核心链路来说，它的重要性就相对低一些。

我们要给电商系统设定 SLO，大的原则就是**先设定核心链路的 SLO，然后根据核心链路进行 SLO 的分解**。接下来，我们就一步步拆解动作，看看怎么实现这个目标。

核心链路：确定核心应用与强弱依赖关系

要确定核心链路的 SLO，我们得先找到核心链路。怎么实现呢？

我们可以先通过全链路跟踪这样的技术手段找出所有相关应用，也就是呈现出调用关系的拓扑图，下面我给了一张图，这个图你应该并不陌生，这是亚马逊电商分布式系统的真实调用关系拓扑图。之所以这么复杂，是因为即使是单条路径，它实际覆盖的应用也是很多的，特别是对于大型的分布式业务系统，可能会有几十甚至上百个路径，这些应用之间的依赖关系也特别复杂。



亚马逊电商分布式系统调用关系拓扑图

防止断更，请添加
首发微信：17716143665

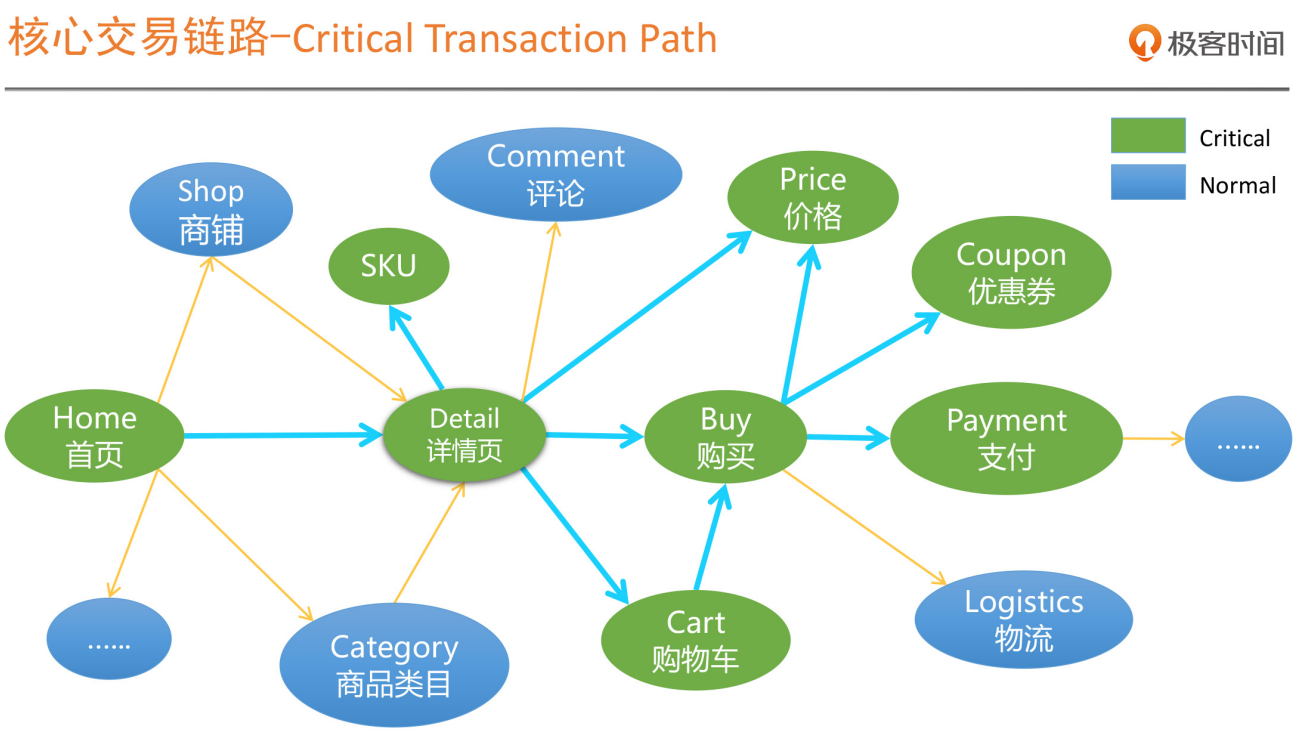
面对这样复杂的应用关系，是不是感觉无从下手？那就继续来做精简，也就是**区分哪些是核心应用，哪些是非核心应用**。这是要根据业务场景和特点来决定的，基本上需要对每个应用逐个进行分析和讨论。这个过程可能要投入大量的人工来才能完成，但这个是基础，虽然繁琐且费时费力，也一定要做。

这里我简单举个例子说明。

比如用户访问商品详情的时候，会同时展现商品评价信息，但是这些信息不展现，对于用户选择商品也不会造成非常大影响，特别是在双十一大促这样的场景中，用户在这个时刻的目的很明确，就是购买商品，并不是看评价。所以类似商品评价的应用是可以降级的，或者短时间不提供服务。那这种不影响核心业务的应用就可以归为非核心应用。

相反的，像商品 SKU 或优惠券这样的应用，直接决定了用户最终的购买金额，这种应用在任何时刻都要保持高可用。那这种必须是高可用的应用就是核心应用。

就这样，我们需要投入一些精力一一来看，确定哪些是核心应用，哪些是非核心应用。梳理完成后，针对电商系统，我们大概可以得到一个类似下图的简化拓扑关系（当然了，这里仍然是示例，相对简化，主要是保证效果上一目了然）。



这张图就呈现出一条典型的电商交易的关键路径，其中绿色部分为核心应用，蓝色部分为非核心应用。

这个时候，我们又要进行关键的一步了，就是确认强弱依赖。

核心应用之间的依赖关系，我们称之为强依赖，而其它应用之间的依赖关系，我们称之为弱依赖，这里就包含两种关系，一种是核心应用与非核心应用之间的依赖，另一种是非核心应用之间的依赖。

好了，至此，我们就确定了这个电商系统的核心应用和强弱依赖关系，也就是找到了这个系统的核心链路。那我们就可以设定这条核心链路的 SLO 了，并基于此设定其他的 SLO。这时应该遵循哪些原则呢？

设定 SLO 有哪些原则？

针对核心和非核心应用，以及强弱依赖关系，我们在设定 SLO 时的要求也是不同的，具体来说，可以采取下面 4 个原则。

第一，核心应用的 SLO 要更严格，非核心应用可以放宽。 这么做，就是为了确保 SRE 的精力能够更多地关注在核心业务上。

第二，强依赖之间的核心应用，SLO 要一致。 比如下单的 Buy 应用要依赖 Coupon 这个促销应用，我们要求下单成功率的 SLO 要 99.95%，如果 Coupon 只有 99.9%，那很显然，下单成功率是达不成目标的，所以我们就会要求 Coupon 的成功率 SLO 也要达到 99.95% 。

第三，弱依赖中，核心应用对非核心的依赖，要有降级、熔断和限流等服务治理手段。 这样做是为了避免由非核心应用的异常而导致核心应用 SLO 不达标。

第四，Error Budget 策略，核心应用的错误预算要共享，就是如果某个核心应用错误预算消耗完，SLO 没有达成，那整条链路，原则上是要全部暂停操作的，因为对于用户来说，他不会判断是因为哪个应用有问题，导致的体验或感受不好。所以，单个应用的错误预算消耗完，都要停止变更，等问题完全解决再恢复变更。当然，也可以根据实际情况适当宽松，如果某个核心应用自身预算充足，且变更不影响核心链路功能，也可以按照自己的节奏继续做变更。这一点，你可以根据业务情况自行判断。

如何验证核心链路的 SLO?

梳理出系统的核心链路并设定好 SLO 后，我们需要一些手段来进行验证。这里，我给你介绍两种手段，一种是容量压测，另一种就是 Chaos Engineering，也就是混沌工程。

容量压测

我们先来看容量压测。容量压测的主要作用，就是看 SLO 中的 Volume，也就是容量目标是否可以达成。对于一般的业务系统，我们都会用 QPS 和 TPS 来表示系统容量，得到了容量这个指标，你就可以在平时观察应用或系统运行的容量水位情况。比如，我们设定容量的 SLO 是 5000 QPS，如果日常达到 4500，也就是 SLO 的 90%，我们认为这个水位状态下，就要启动扩容，提前应对更大的访问流量。

容量压测的另一个作用，就是看在极端的容量场景下，验证我们前面说到的限流降级策略是否可以生效。

我们看上面电商交易的关键路径图，以 Detail（商品详情页）和 Comment（商品评论）这两个应用之间的弱依赖关系为例。从弱依赖的原则上讲，如果 Comment 出现被调用次数过多超时或失败，是不能影响 Detail 这个核心应用的，这时，我们就要看这两个应用之间对应的降级策略是否生效，如果生效业务流程是不会阻塞的，如果没有生效，那这条链路的成功率就会马上降下来。

另外，还有一种场景，如果某个非核心应用调用 Detail 的次数突然激增，对于 Detail 来说，它自身的限流保护机制要发挥作用，确保自己不会被外部流量随意打垮。

其实类似上述这两种场景，在分布式系统中仅仅靠分析或画架构图是无法暴露出来的，因为业务变更每天都在做，应用之间的调用关系和调用量也在随时发生变化。这时候就需要有容量压测这样的手段来模拟验证，进而暴露依赖关系问题。并且，有些问题必须要在极端场景下模拟才能验证出问题，比如各种服务治理措施，只有在大流量高并发的压力测试下，才能被验证出是否有效。

Chaos Engineering- 混沌工程

好了，接下来看第二种混沌工程。

我们知道，现在混沌功能非常流行，因为它可以帮助我们做到在线上模拟真实的故障，做线上应急演练，提前发现隐患。很多公司都很推崇这种方法并在积极学习落地中。

其实，刚才我们讲容量压测也提到，容量压测是模拟线上真实的用户访问行为的，但是压测过程中，如果我们模拟极端场景，可能也会造成异常发生，但这时的异常是被动发生的，不过从效果上来讲，其实跟混沌工程就很相似了。只不过，混沌工程是模拟故障发生场景，主动产生线上异常和故障。

那我们怎么把混沌工程应用在 SRE 中呢？一般来说，故障发生的层面不同，我们采取的方式也会不同。

这里我简单介绍一下，比如对于机房故障，有些大厂会直接模拟断电这样的场景，看机房是否可以切换到双活或备用机房；在网络层面，我们会模拟丢包或网卡流量打满；硬件和系统

层面，可能故意把一块磁盘写满，或者把 CPU 跑满，甚至直接把一个服务器重启；应用层面，更多地会做一些故障注入，比如增加某个接口时延，直接返回错误异常，线程池跑满，甚至一个应用集群直接下线一半或更多机器等。

其实混沌工程也是一个非常复杂的系统化工程，因为要在线上制造故障，或多或少都要对线上业务造成影响，如果模拟故障造成的真实影响超过了预估影响，也要能够快速隔离，并快速恢复正常业务。即使是在稳定性体系已经非常完善的情况下，对于混沌工程的实施也要极为谨慎小心。对于一个模拟策略上线实施，一定是在一个隔离的环境中经过了大量反复验证，包括异常情况下的恢复预案实施，确保影响可控之后，在经过多方团队评审或验证，才最终在线上实施。

从这个角度来讲 SRE 和混沌工程是什么关系，就非常清晰了，**混沌工程是 SRE 稳定性体系建设的高级阶段，**一定是 SRE 体系在服务治理、容量压测、链路跟踪、监控告警、运维自动化等相对基础和必需的部分非常完善的情况下才会考虑的。

所以，引入混沌工程手段要非常慎重，我建议大可不必跟风过早引入，还是优先一步步打好基础再做考虑。

应该在什么时机做系统验证?

我们有了验证整个系统 SLO 的手段，但是我们可以看到，这两个手段都是要在生产系统上直接实施的，为了保证我们的业务正常运行，那我们应该选择在什么时机，以及什么条件下做系统验证呢?

我们可以参考 Google 给出的建议。

核心就是错误预算充足就可以尝试，尽量避开错误预算不足的时间段。因为在正常业务下，我们要完成 SLO 已经有很大的压力了，不能再给系统稳定性增加新的风险。

同时，我们还要评估故障模拟带来的影响，比如，是否会损害到公司收益？是否会损害用户体验相关的指标？如果造成的业务影响很大，那就要把引入方案进行粒度细化，分步骤，避免造成不可预估的损失。

这里，我和团队通常的做法，就是选择凌晨，业务量相对较小的情况下做演练。这样即使出现问题，影响面也可控，而且会有比较充足的时间执行恢复操作，同时，在做较大规模的全

站演练前，比如全链路的压测，会做单链路和单业务的单独演练，只有单链路全部演练通过，才会执行更大规模的多链路和全链路同时演练。

总之，生产系统的稳定性在任何时候，都是最高优先级要保证的，决不能因为演练导致系统异常或故障，这也是不被允许的。所以，一定要选择合适的时机，在有充分准备和预案的情况下实施各类验证工作。

总结

今天我们结合一个电商案例，讨论了在落地 SLO 时还要考虑的一些实际因素。你需要重点掌握以下 4 点：

1. 先设定核心链路的 SLO，然后根据核心链路进行 SLO 的分解。
2. 确定一个系统的核心链路，关键是确认核心应用和非核心应用，以及强弱依赖关系。这个过程在初始执行时，需要大量人工投入和分析，但是这个过程不能忽略。
3. 强依赖的核心应用 SLO 必须与核心链路 SLO 一致，弱依赖的非核心应用 SLO 可以降低，但是必须要有对应的限流降级等服务治理手段。
4. 通过容量压测和混沌工程等手段，对上述过程进行验证，但是一定要在错误预算充足的情况下执行。

思考题

最后，给你留一个思考题。

我们知道，不同的业务场景，考虑的限制因素也是不同的，对 SLO 设定也是一样的。本节课我们分享了一个电商业务应该要考虑的因素，你是不是可以按照这个思路，分享一个不同于电商的业务场景，应该或可能要考虑哪些特殊因素呢？

期待在留言区看到你的思考，也欢迎你把今天的内容分享给身边的朋友。

我是赵成，我们下节课见。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　04 | 错误预算：达成稳定性目标的共识机制

下一篇　06 | 故障发现：如何建设On-Call机制?

精选留言 (5)

写留言

lyonger
2020-03-28

期待老师分析全链路跟踪的相关实践。

展开

作者回复: 可以看我第一门课里相关的章节，也可以自己在其它专栏中找一下对应的内容。因为这部分内容讲的比较多了，我就不再这个专栏中赘述了。

1　1

自行车
2020-04-01

很有启发，为公司部署了很多监控系统和告警策略，但是总感觉大部分的告警都是无用的，现在想想就是没有制定明确的slo，感谢老师

展开

作者回复: less is more，一定要选择有用的和关键的指标

leslie
2020-03-30

老师在课程中讲到了生产压测，其实压测同样可以放在测试环境或者非核心业务中去测试；这其实就是结合DevOps方面的知识。之前学习时有被推荐此书，不过精力有限尚未来的及去增加书库。
DevOps/敏捷讲的是突出效果必然要用典型系统：SRE的不少操作完全可以反其道而行之；故而个人意志觉得DevOps和SRE是互补的，如何合理使二者发挥功效这其实是我们...
展开

作者回复: 感谢你的分享和建议。

测试环境也可以做压测，特别是一些大型核心系统，做线上压测时可以在测试环境上提前做几次，先看下效果。

但是测试环境最大的问题是，数据量是没有线上那么大，模型也没有那么精准的，所以最有效的办法还是线上。

关于DevOps和SRE的关系，后续我会在答疑篇中分享一下。

摇滚诗人M
2020-03-29

游戏（吃鸡）业务：
1.核心链路：登录，玩家组队及匹配，道具购买（非核心链路，但是对公司收益直接影响，故也应保障），进行游戏，游戏结算（经验值等）。
2. 除了valet维度的SLO，还需招募人员，内测游戏中的各种场景下可能出现的bug。（bug budget）...
展开

作者回复: 很好的回顾了我们本节课程的内容，学以致用。

Christopher

2020-03-29

给力，运维需要系统的学习

展开